Item 1/20

When a module is imported, its contents:

- are executed as many times as they are imported
- are ignored
- may be executed (explicitly)
- **are executed once (implicitly)**

Next →

Back to Summary

Item 2/20

UNICODE is a standard:

- (●) like ASCII, but much more expansive
- ( ) honored by the whole universe
- ( ) used by coders from universities
- ( ) for coding floating-point numbers

← Prev  Next →

Back to Summary

Item 3/20

The top-most Python exception is named:

- `PythonException`
- `TopException`
- (•) `BaseException`
- `Exception`

← Prev  Next →

Back to Summary

Item 4/20

Knowing that a function named `fun()` resides in a module named `mod`, choose the proper way to import it:

- `from fun import mod`
- `import fun`
- `import fun from mod`
- (●) `from mod import fun`

← Prev  Next →

Back to Summary

Item 5/20

A function returning a list of all entities available in a module is called:

- `content()`
- `entities()`
- (•) `dir()`
- `listmodule()`

← Prev  Next →

Back to Summary

Item 6/20

The following code:

```
print(3 * 'abc' + 'xyz')
```

prints:

- abcabcxyzxyz
- xyzxyzxyzxyz
- abcabcabcxyz
- abcxyzxyzxyz

← Prev  Next →

Back to Summary

Item 7/20

ASCII is:

- (●) short for *American Standard Code for Information Interchange*
- ( ) a standard Python module name
- ( ) a predefined Python variable name
- ( ) a character name

← Prev  Next →

Back to Summary

Item 8/20

A predefined Python variable that stores the current module name is called:

- (●) `__name__`
- ( ) `__module__`
- ( ) `__mod__`
- ( ) `__modname__`

← Prev  Next →

Back to Summary

Item 9/20

UTF-8 is:

- a Python version name
- a synonym for *byte*
- the 9th version of the UTF standard
- a form of encoding Unicode code points

← Prev Next →

Back to Summary

Item 10/20

The following code:

```
x = '\''
print(len(x))
```

prints:

- 20
- 3
- 2
- 1 ●

← Prev Next →

Back to Summary

# Item 11/20

The following statement:

```
from a.b import c
```

causes the import of:

- entity `c` from module `a` from package `b`
- (●) entity `c` from module `b` from package `a`
- entity `b` from module `a` from package `c`
- entity `a` from module `b` from package `c`

← Prev  Next →

Back to Summary

Item 12/20

Knowing that a function named `fun()` resides in a module named `mod`, and it has been imported using the following line:

```
import mod
```

Choose the way it can be invoked in your code:

- `fun()`
- `mod::fun()`
- `mod->fun()`
- (●) `mod.fun()`

← Prev  Next →

Back to Summary

Item 13/20

The following code:

```
print(float("1, 3"))
```

- raises a `ValueError` exception
- prints `13`
- prints `1,3`
- prints `1.3`

← Prev  Next →

Back to Summary

Item 14/20

The following code:

```
print(ord('c') - ord('a'))
```

prints:

- [x] 2
- [ ] 0
- [ ] 3
- [ ] 1

← Prev  Next →

Back to Summary

Item 15/20

The following code:

```
print('Mike' > "Mikey")
```

prints:

- (●) False
- ( ) 1
- ( ) True
- ( ) 0

← Prev  Next →

Back to Summary

Progress (100%)

Item 16/20

The unnamed `except:` block:

- can be placed anywhere
- must be the first one
- cannot be used if any named block has been used
- **must be the last one**

← Prev
Next →

Back to Summary

Item 17/20

Entering the `try:` block implies that:

- the block will be omitted
- some of the instructions from this block may not be executed
- all of the instructions from this block will be executed
- none of the instructions from this block will be executed

← Prev Next →

Back to Summary

Item 18/20

The `pyc` file contains:

- (●) compiled Python code
- ( ) a Python compiler
- ( ) Python source code
- ( ) a Python interpreter

← Prev  Next →

Back to Summary

Item 19/20

The following code:

```
print(chr(ord('z') - 2))
```

prints:

- (●) x
- ( ) z
- ( ) y
- ( ) a

← Prev  Next →

Back to Summary

Item 20/20

The following statement:

```
assert var == 0
```

- will stop the program when `var == 0`
- is erroneous
- (●) will stop the program when `var != 0`
- has no effect

Prev

Back to Summary